Report on the CNN Code
PD-Internship
Lerner Paul
24/07/2018

Inspired by the works of Oh et al. and Bai et al. (cf. Report #3 and #4 respectively), I tried to use 1d CNN instead of a RNN.
Our work is then similar to Pereira et al. as we interpret the sensors signals as an image but instead of resizing it to a 2d, square image we keep it 1d. Moreover we keep the measures as an "image channel" dimension instead of having *m* columns for *m* measures as them (cf. report #2). Also, as them, we perform deep, end-to-end learning (we're the only two to do so on PD diagnosis via HaW, to the best of my knowledge, cf. report #2). Update : now catherine has done it, should we cite her even if her paper hasn't been published ?

Disambiguation : unlike in the Report on RNN where I tried different evaluation techniques, here early stopping is never performed, I always train the model for 100 epochs then give the results of 10-CV accuracy after choosing the best epoch **(for all 10 folds at once)**.

Note : I used to refer as the sentence task as "*tram* task" but I will now refer to it as "sentence task" to be consistent with Moetesum et al. and as "tram" is not the whole writing in the task, unlike in e.g. the *lektorka* task.

Contents

1 Model architecture

Hypothesis :

1. As a 2d conv takes the spatial aspect of the data into account, a 1d conv takes the sequential aspect of the data into account
2. We don't need to predict a sequence so we don't need to pad the CNN layers or use causal convolutions as Bai et al. (cf. Report #4)
3. dilated convolutions allow the model to have a "large scope" before taking a decision (see Bai et al.).
4. a deep network will a priori extract advanced/complex features and we have don't have enough data for that.
5. we should aim for a feature map between 5 and 200 neurons, it's the number of features used by Mucha et al. and Drotar et al., respectively (cf. report #2)
6. stride is only good to reduce the dimensionality of the data/features
7. using a strided convolution or a strided pooling seems to be equivalent, therefore, intuitively, I prefer to use strided pooling and keep the stride of conv layer(s) at 1.

Questions :

1. how large should the kernel(s) be ?
2. should we first have a high dimensional data (i.e. with numerous filters) that we then pool or start with a small number of filters ?
3. how dilated should the conv(s) be ? Do the network needs to have a large scope ? I will first experiment without dilation.

In order to have the same sequence length on all samples I padded the spirals to 16071 timesteps (max. length). We could also keep the data untrimmed and unpadded then pool with a kernel of the same size as the sequence length dimension in order to have only one neuron per filter. That's what I do in the CNN-baseline, see below.

1.1 Vanilla CNNs

1.1.1 CNN100-baseline

- only 1 conv layer with 100 kernels of size 1 (thus stride and dilation of 1) ; ReLU activation and weight_norm regularization
- one pooling layer with a filter of 16071 (max length of the spiral) thus reducing the sequence to 1.
- dropout with probability 0.2
- one FC layer (fed the feature map of size 100) with a sigmoid activation

Later I might refer to this model as CNN***-baseline with *** being the number of kernels.

It's curious that we achieve decent results on all tasks with the CNN50-baseline (see 2 Results on multiple tasks) although we're not taking into account the sequential aspect of the data as we use a kernel size of 1.

1.1.2 Random Search

After padding all the spirals to 16071 (max. length), I performed a quick random search (~100 experiments) to find hyperparameters for a **2 layer CNN** in the following sets :

- learning_rate : {0.001, 0.0001}
- dropout : {0, 0.1, 0.2, 0.3, 0.4, 0.5}
- hidden_size : {4, 8, 16, 32}
- dilation : {1, 2, 4, 8, 16}
- conv_kernel : {1, 2, 4, 8, 16}
- pool_kernel : {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}

I achieved ~ 70% accuracy on the spiral task with 4 quite different models :

Table. Model hyperparameters displayed as [layer1, layer2].

learning rate	hidden_size	conv_kernel	pool_kernel	dilation	dropout
0.001	[4, 16]	[16, 2]	[512, 15]	[4, 16]	0
0.0001	[4, 32]	[2, 8]	[128, 118]	[8, 1]	0.1
0.001	*[4, 32]*	*[16, 16]*	*[64, 128]*	*[2, 2]*	*0.3*[1]

[1] Since this model provided interesting results, I baptized it **CNN-Two** and used it to evaluate Data augmentation, see 3.1 Augmentation

0.001	[16, 32]	[8, 8]	[32, 474]	[1, 4]	0

This does not support hypothesis #4, we might extract deep features from the data. It's quite hard to interpret this results, we can conclude that there's a lot of interactions between the different hyperparameters. Also the architecture is quite robust, most of the other "randomly" parametrized achieved more than 60% accuracy. Also we can see that none of these models has a convolutional kernel of size 1 as CNN***-baseline, although it achieves better results. This suggests that one should take the sequential aspect of the data into account.
However these models achieve poor performance on other tasks like l and lektorka so I performed another random search for a **1 layer CNN** on the spiral task. The results reveal that the pool_kernel should be quite large, see figure below. We cannot make similar observations for the other hyper parameters as there is a lot of interaction between them (e.g. between convolutional kernel size and dilation). From this random search came out one model that turned out to outperform the baseline on most tasks, see 2 Results on multiple tasks. I called it **CNN-One** as it has only one layer.

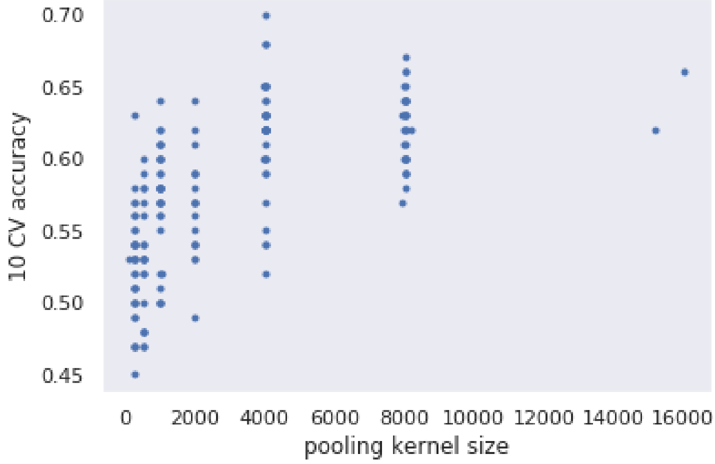


Fig. 10 CV accuracy vs pooling kernel size over 170 experiments on the spiral task

CNN-one is a very small network with only 16 hidden units. It's kernel is only of size 2 but has a dilation of 16 which should allow for a quite large scope. The result of the convolution is pooled so its length is only four. A dropout of 0.2 is applied before feeding the feature map to the FC layer.

1.1.3 Splitting data into strokes

1.1.3.1 Using 2 separate models for in-air and on-paper strokes

In order to provide a baseline for the Hierarchical models (cf. 1.2), especially the HS CNN and H CRNN, I trained a CNN-one on all strokes of a task independently before applying majority voting to merge the strokes predictions by task (i.e. subject). Two separate models are trained on in-air and on-paper strokes, respectively, in the same way as the first layer of HS CNN and H CRNN.
The CNN-one provides a strong baseline, see 2.2.

As this experiment suggests that splitting data into strokes augments the dataset, I performed another random search using the same sets as before, although this time the model might have up to 4 layers (another hyperparameter chosen randomly). This search was performed on the *porovnat* task. After 134 experiments, we're not able to find any statistical difference depending on the model learning rate, dropout or number of layers. This goes along with the previous results which shows that there is a lot of interaction between the different hyperparameters. Moreover, concerning the learning rate, I previously noted that both 10^{-3} and 10^{-4} were suited for most models. This is in line with the results of Greff et al. on LSTMs which demonstrate that the learning rate is dependent on the dataset but that, independently of the dataset, there is a large basin (up to two orders of magnitude) of good learning rates inside of which the performance does not vary much (cf report #4). From this search came out a quite strange model with three layers that I baptized CNN-Three. I compare it with CNN-One in 2.1.

CNN-Three is trained with a learning rate of 10^{-4} and is made of three convolutional blocks :

1) Conv1d(7, 128, kernel_size=16, dilation=16)
   MaxPool1d(kernel_size=8)
2) Conv1d(128, 32, kernel_size=4, dilation=16)
   MaxPool1d(kernel_size=16)
3) Conv1d(32, 128, kernel_size=1, dilation=1)
   MaxPool1d(kernel_size=1)

A dropout of 0.3 is applied after pooling. Note that the last pooling layer is ineffective. The kernel size of the 2 first pooling layers is dependent on the task.

1.1.3.2 Using a single model for both in-air and on-paper strokes

In order to study the effect of using a separate model for in-air and on-paper strokes, I trained CNN-One and CNN-Three on all strokes regardless of the button status. Results and analysis is available in 2.1.

1.2 Hierarchical Models

In order to augment the dataset, I tried splitting the tasks into subsequences : either stroke or "token" (e.g. "le", "les", see below). These models all have the same default : they're not applicable to the spiral task, although we might use only the first hierarchical block for it (see below).

1.2.1 Hierarchical CNN

As with RNN, I tried to build a hierarchical CNN on the *l* task : I split the task into letters (i.e. *l*s). See figure below. Unlike the previous RNN experiment, here I individually standardized each letter. This should give to the model the ability to treat each stroke as a training example, if we were to standardize the whole task instead, the x-coordinate would be higher

and higher for each stroke (as Czechs write from left to right). Moreover, I padded each letter to 1242 timesteps (max. length). Since the subjects n° 12, 21, 23, 44, 67 did 6 l instead of 5 l discarded their last l.
Some subjects had on-paper strokes between the ls so I concatenated those with the previous closest l in order to maintain 5 ls per subjects.
See 1.2.2 for architecture figure.

With this Hierarchical CNN we're able to achieve 65% accuracy on 10-CV. This is the first time we're able to go above chance level with the l task.
The hyperparameters were found with a quick random search (~10 experiments) in the same sets as before except for the pool_kernel which was in $\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$.

The same experiment was conducted with the le task (split into 5 "le"). Results are very similar. Only difference between the models is that Hierarchical used a Pool1 of 128 instead of 32 in order to keep the output length to the same order of magnitude as with the l task.

The inconvenient with this model is that it's not applicable to all tasks. Moreover, it's hard to tell where to split the task into letters or in sub-task.
I compared this model with an equivalent CNN (**e2h**) which was fed the whole l task. The model is the same except that the input is whole l task padded to 4226 timesteps (max length) so there's no concatenation obviously. Moreover, in order to provide roughly the same number of parameters, The Pool1 has a stride of 24 instead of 32 so the output length would be 173, as close as possible to 180.

This equivalent model achieves 58% accuracy on 10 CV so it suggests that the letter split augments the dataset and allows the hierarchical model to generalize better than the simple model. See table below.
I conducted the same experiment on the spiral task, using a Pool1 of stride 80 instead of 32 so the output length would be as close as possible to 180. It provided similar results as with the l task. The Hierarchical and the e2h model have too much parameters for this amount of training data. Dropout of more than 0.1 has made the training unstable on several models.

I also tried to learn over the spiral task with the letter-level block of the Hierarchical model. Just using a Pool1 of stride 8000 instead of 32 so that the FC layer still had 32 features as an input.

Table. Experiment results w.r.t. architecture. Best results per task are print in bold.

Task	***Model***	***Training acc***	***Test acc***
spiral	**CNN50-baseline**	0.74 (± 0.04)	**0.67** (± 0.15)
l (whole task)	CNN50-baseline	0.65 (± 0.06)	0.60 (± 0.12)
l (5 letters)	**Hierarchical CNN**	0.98 (± 0.02)	**0.65** (± 0.10)

l (whole task)	e2h CNN	0.93 (± 0.04)	0.58 (± 0.09)
spiral	e2h CNN	0.97 (± 0.02)	0.60 (± 0.17)
spiral	Hierarchical letter-level block	0.88 (± 0.02)	0.60 (± 0.11)

1.2.2 Hierarchical Stroke CNN (HS CNN)

As you'll see in 5.1.2 on-paper and in-air strokes, I hypothesized that the hierarchical model (and by extension, all CNNs) used the same filters to monitor both on-paper and in-air strokes. However, we know that both are very different, so I assumed they disturbed the model. Thus I extended Hierarchical CNN to Hierarchical Stroke CNN (HS CNN) where there is two *Conv1* :

1. for the on-paper strokes
2. for the in-air strokes

In order to maintain the number of strokes to 9 (5 ls + 4 in-air strokes between) among all subjects I discarded the strokes that were previously concatenated (see 1.2.1 Hierarchical CNN).

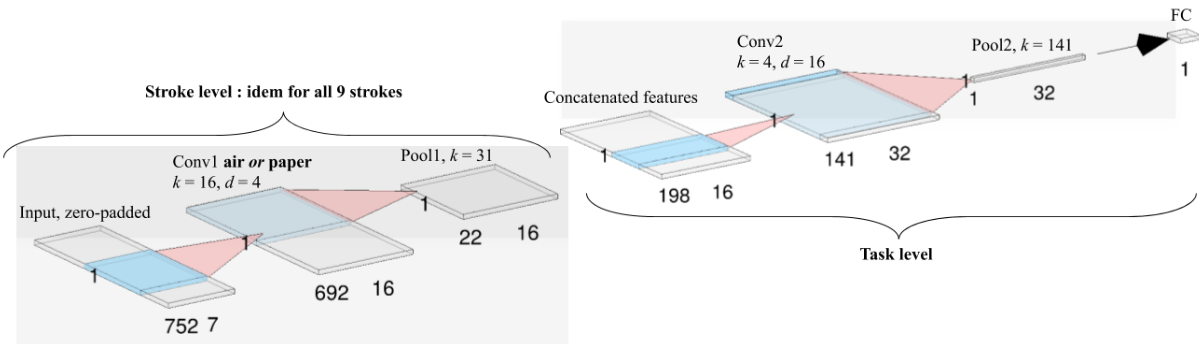


Fig. HS CNN. A dropout of 0.1 is applied after the pooling layers. Both Conv layers have ReLU activation and weight norm regularization. FC layer has Sigmoid activation. *k* stands for kernel size and *d* stands for dilation. Note that Conv1 is either Conv1_air or Conv1_paper and gets fed in-air and on-paper strokes, respectively.

This greatly improved the results over the l task (see table below). Similar results have been obtained with le and les task. This supports my hypothesis that the model doesn't use the button_status measure to its full advantage.
However it's quite difficult to apply this model to other tasks because they contain a lot of "intra-word" stroke which are not the same among all subjects so it's hard to tell which stroke should be discarded (see below).

In order to resolve this problem we could :

1. use all strokes and pad the strokes' list with a matrix of zeros.
2. use a RNN instead of a CNN for the task-level block of HS CNN.

Table. Experiment results w.r.t. hierarchical architecture. Best results are print in bold.

Task	*Model*	*Training acc*	*Test acc*
l (5 letters)	Hierarchical CNN	0.98 (± 0.02)	0.65 (± 0.10)
l (9 strokes)	**HS CNN**	0.98 (± 0.02)	**0.77** (± 0.16)
l (zero-padded to 15 strokes)	HS CNN	0.96 (± 0.03)	0.75 (± 0.11)
majority voting on zero-padded l, le and les tasks	HS CNN	1.00 (± 0.00)	**0.78** (± 0.13)

I tried the first point and it provided similar results on the l, le and les tasks (see table above). However, it doesn't work very well with the lektorka and the sentence task. This may be because the # of strokes is highly variable on these tasks, unlike in the l, le and les tasks. E.g. on the le task, most strokes will be either on-paper "le" or in-air movement between 2 "le"s. However, on, e.g. lektorka, depending on the subject, the strokes might be "lektorka", "lekt", or "l".

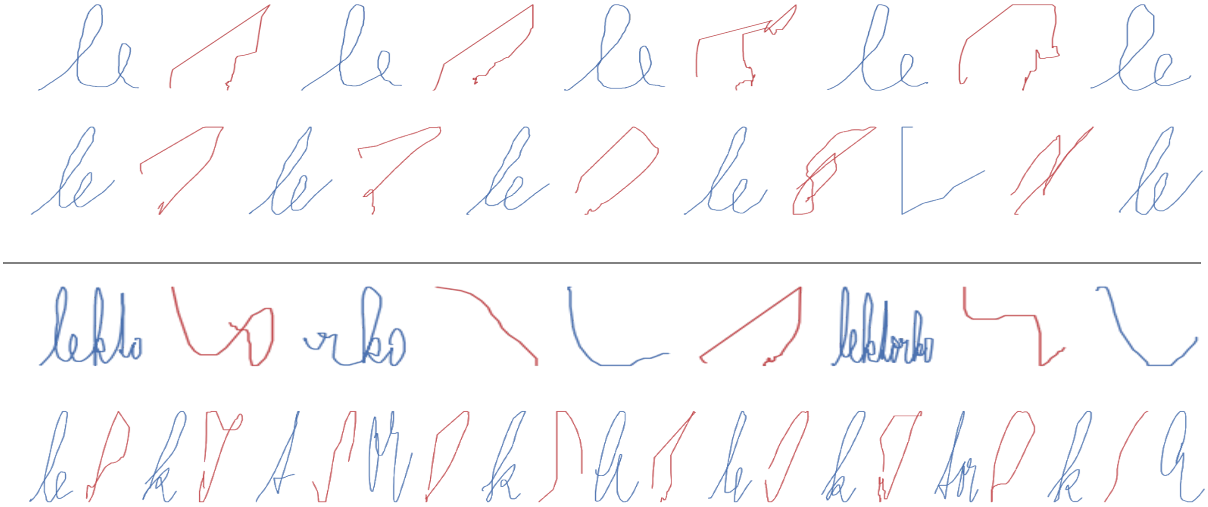
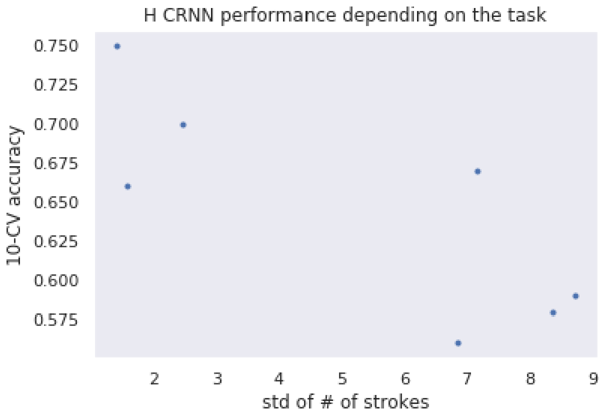


Fig. On-paper (blue) and in-air (red) strokes for subjects 70 (control, above) and 3 (PD, below) on tasks le (above) and lektorka (below). Note that the data has been standardized and that each stroke is displayed at its own scale.

I describe the second point in the next section.

1.2.3 Hierarchical Convolutional-Recurrent (H CRNN)

As splitting the strokes into air and paper improved the results over the H CNN, I assume that it would also for this model and didn't try to use full strokes.
As I explained in the last section, the HS CNN is hard to apply on most tasks which have a great varying number of strokes. So I replaced the task-level block of HS CNN with a RNN (On the lektorka task it provided similar results with a GRU and a LSTM). Thus there's no need to pad the strokes nor the list of strokes. The RNN is simply fed the concatenated features of length y (depending on the strokes' length) of the x strokes.
Surprisingly, the model achieves good results on the l task, unlike fully LSTM or GRU models we used previously. This may be because dependencies between timesteps are too long for a RNN. I again performed a random search on the l task and found a quite similar model (see figure below), as using the same hyperparameters as HS CNN led to overfitting (a LSTM with input of size 16 and hidden state of size 32 has 4352 parameters). The model is trained with a learning rate of 10^{-4} instead of 10^{-3}.



However, as you'll see in 2 Results on multiple tasks, the model doesn't solve the problem of HS CNN and the results are still quite poor on tasks like lektorka, nepopadnout, porovnat and the sentence. After a statistical analysis using Pearson correlation coefficient, I found that the model performance was negatively correlated with the std of the number of strokes per task with a p-value of 0.03 (strongly significant), see figure.

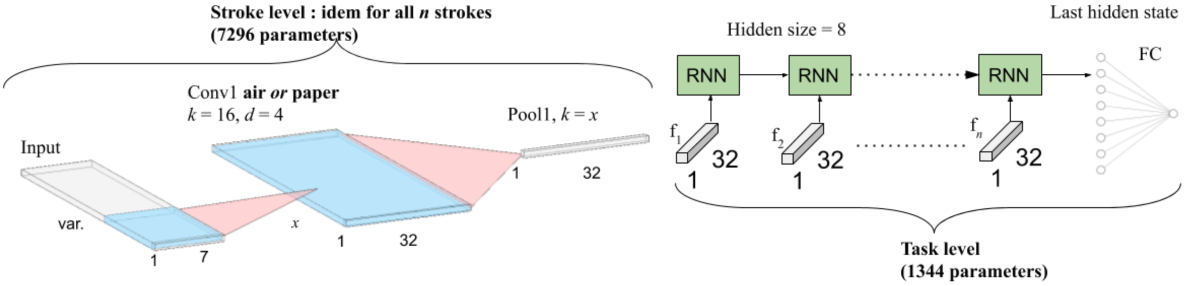


Fig. H CRNN. The kernel / stride of the Pooling layer is different between the tasks. It is set so that the result has a length of 1.

I said that there's no need to pad the strokes but that's an abuse of language because previously all strokes had to have the same length. However, we still have to pad the very short strokes to 61 timesteps as the first convolutional layer has a kernel size of 16 and a dilation of 4.

$$\boxed{min = 1 + dilation \times (kernel - 1)}$$

This approach is similar to Catherine's and also Chiu & Nichols who extracts character-level features from character embeddings using a CNN before feeding the feature vectors to a LSTM in order to perform Named Entity Recognition (NER), cf report #4. It's also similar to Granell et al. who slide a CNN which extracts features over a document before feeding those features to a LSTM for offline handwriting recognition.
This kind of model may be useful when the input data is very long, I think that the CNN layer acts as kind of dimensionality reduction, but I'm only conjecturing from our previous experiments.

See results in the 2 Results on multiple tasks. Moreover, using a CNN equivalent to the first block I achieved 64% accuracy on the spiral task.

2 Results on multiple tasks

2.1 CNN-One VS CNN-Three and Single VS Double models

Note all these models were trained on a given task split into strokes and had their prediction fused to get a task-level (i.e. subject-level) prediction. "single" models are trained on both in-air and on-paper strokes while "double" models are specific to both respectively, cf . 1.1.3 Splitting data into strokes.

10-CV accuracy (± std) w.r.t. model architecture

task / model	***double CNN-One***	***Single CNN-One***	***double CNN-Three***	***single CNN-Three***
spiral	0.63 (± 0.14)	0.65 (± 0.16)		
l	0.70 (± 0.23)	0.65 (± 0.21)	0.75 (± 0.12)	**0.80** (± 0.12)
le	0.72 (± 0.14)	**0.75** (± 0.16)	0.66 (± 0.10)	0.70 (± 0.17)
les	**0.72** (± 0.13)	0.71 (± 0.12)	0.68 (± 0.11)	0.70 (± 0.11)
lektorka	0.62 (± 0.15)	0.63 (± 0.16)	0.61 (± 0.15)	0.59 (± 0.14)
porovnat	0.62 (± 0.12)	0.60 (± 0.12)	**0.71** (± 0.08)	0.66 (± 0.16)
nepopadnout	0.67 (± 0.11)	**0.68** (± 0.09)	**0.68** (± 0.16)	
sentence	0.62 (± 0.07)	0.61 (± 0.14)	**0.63** (± 0.23)	
all tasks	**0.70** (± 0.10)	0.69 (± 0.12)		

We can see that double and single models provides similar results. Therefore, the difference between the performance of H CNN and HS CNN (see 1.2.2) might be explained because of the representation of the data in H CNN : the on-paper and in-air strokes were concatenated. When you think about it, it's not that surprising that the model extracts the same features regardless of the button status (see 5.1.2) as Drotar et al. use the same hand-crafted features for both on-paper and in-air strokes.
It's hard to tell between CNN-One and CNN-Three, the experiment is on going...

2.2 Late fusion VS Hierarchical models

10-CV accuracy (± std) w.r.t. model architecture

task / model	***H CRNN***	***double CNN-One (strokes)***
spiral	NA	0.63 (± 0.14)
l	**0.75** (± 0.11)	0.70 (± 0.23)
le	0.66 (± 0.08)	**0.72** (± 0.14)
les	**0.70** (± 0.15)	**0.72** (± 0.13)
lektorka	0.67 (± 0.13)	0.62 (± 0.15)
porovnat	0.59 (± 0.15)	0.62 (± 0.12)
nepopadnout	0.56 (± 0.11)	0.67 (± 0.11)
sentence	0.58 (± 0.14)	0.62 (± 0.07)
all tasks[2]	0.73 (± 0.13)	0.70 (± 0.10)

CNN-one provides a strong baseline for the H CRNN, moreover it's applicable to all tasks unlike H CRNN. Also the results are more consistent over the tasks (see e.g. nepopadnout). However, it still falls slightly below H CRNN after majority voting over all tasks.

2.3 Our main models VS State of the art

Note that our models were not trained with the same number of epochs on all tasks, thus our results are a bit overoptimistic (between 1-5% accuracy). I don't know if we should display the results with models trained with the same n° of epochs on all tasks.

[2] But spiral on the H CRNN

10-CV accuracy (± std) compared to Drotar et al. and Moetesum et al.

task / model	***CNN50 baseline***	***CNN-One (whole task)***	***CNN-One (strokes)***	***H CRNN***	***Drotar et al.[3]***	***Moetesum et al.***
spiral	0.67 (± 0.15)	**0.68** (± 0.16)	0.63 (± 0.14)	NA	0.63	**0.76** ± 0.08
l	0.60 (± 0.12)	0.60 (± 0.14)	0.70 (± 0.23)	**0.75** (± 0.11)	0.72	0.62 ± 0.08
le	0.62 (± 0.17)	0.62 (± 0.17)	**0.72** (± 0.14)	0.66 (± 0.08)	0.71	0.57 ± 0.09
les	0.60 (± 0.12)	0.65 (± 0.14)	**0.72** (± 0.13)	0.70 (± 0.15)	0.66	0.60 ± 0.08
lektorka	0.60 (± 0.20)	**0.68** (± 0.13)	0.62 (± 0.15)	0.67 (± 0.13)	0.65	0.60 ± 0.07
porovnat	0.62 (± 0.15)	0.60 (± 0.13)	0.62 (± 0.12)	0.59 (± 0.15)	**0.73**	0.51 ± 0.09
nepopadnout	0.58 (± 0.16)	0.54 (± 0.12)	0.67 (± 0.11)	0.56 (± 0.11)	**0.68**	**0.68** ± 0.0
sentence	0.63 (± 0.09)	0.52 (± 0.14)	0.62 (± 0.07)	0.58 (± 0.14)	**0.76**	0.51 ± 0.08
all tasks[1]	0.60 (± 0.13)	0.63 (± 0.15)	0.70 (± 0.10)	0.73 (± 0.13)	0.81	**0.83** ± 0.09

CNN-One slightly outperforms CNN-50 baseline on most tasks and after majority voting which suggests that it's better to take the sequential aspect of the data into account.
Splitting the data into strokes improves performance over all tasks using CNN-one expect on the spiral task which should be expected as most of the subject performed a single stroke, thus, the few in-air stroke are probably disturbing the model. Although it's not applicable to the spiral task, H CRNN provides the best classification accuracy among our models after majority voting (if we don't take in consideration HS CNN, see 1.2.2 Hierarchical Stroke CNN). This suggests that splitting the data into strokes effectively augments the dataset thus the generalization capacity of the model.
We achieve competitive results with Drotar et al. on the l, le and les tasks with the H CRNN and CNN-One (and we outperform them with the HS CNN, see 1.2.2 Hierarchical Stroke CNN).
We achieve competitive results with Moetesum et al. with all our models, except on the spiral task. Moreover, majority voting results are disappointing. I also tried to fuse the probability outputs of the models instead of majority voting but it provided similar results.

3 Data

3.1 Augmentation

Cf. report on the RNN code.

[3] Drotar et al. don't provide for the std of the 10 CV accuracy

Since CNN-baseline doesn't take the sequential aspect of the data into account, I don't think there's any point applying data augmentation to it.
CNN-One doesn't overfit much as it has only ~300 parameters so I expect that data augmentation will not improve the results a lot.

> *"A key concept of data augmentation is that the deformations applied to the labeled data do not change the semantic meaning of the labels." — Salamon & Bello*

This "key concept" is not as straightforward in our case as in, e.g. ASR or computer vision, a big spiral is still a spiral but is it still a spiral drawn by a PD ?

Table. Summary of the data augmentation techniques

Technique	***Author***	***Applied domain***	***Applicable to (measures)***	***Hyperparameter s***
translation	-	-	All but button_status	Float sampled from U(-0.5, 0.5), measures to apply
rotations	-	-	All but button_status	Random degree (int) in {-15, 15, 30}
speed	Peddinti et al. ; Salamon & Bello	audio / speech	All	resampling factor in {0.5, 0.75, 2}
slice aka clipping aka cropping	Varol et al. ; Sun et al. ; Le guennec et al.	video, time series	All	random int in [[task_len/10 ; task_len/5)]], beginning, end or both
window warping	Um et al.	acceleration	All	resampling factor in {0.5, 0.75, 2}, random spot in the task of random length in [[task_len/10 ; task_len/5)]]
flip	Um et al. ; Wang et al.	acceleration, video	spatial coordinates	axis : horizontal, vertical or both

homothety aka scale	Um et al. ; Peddinti et al.	acceleration, speech	All but button_status	random float in [0.8; 1.2], measures to apply
magnitude warping	Um et al.	acceleration	All but button_status and timestamp	? (mail)

Table. 10-CV accuracy on the spiral task w.r.t. data augmentation and model.

Technique	***CNN-One***[4]	***CNN-Two***	***e2h***
no augmentation (baseline)	*0.68 (± 0.16)*	*0.70 (± 0.20)*	*0.60 (± 0.17)*
translation	0.65 (± 0.16)	0.65 (± 0.16)	?
rotations	0.64 (± 0.18)	0.63 (± 0.24)	0.59 (± 0.19)
slice aka clipping aka cropping	0.66 (± 0.15)	0.62 (± 0.19)	0.60 (± 0.22)
flip	0.62 (± 0.12)	0.60 (± 0.19)	0.58 (± 0.23)
homothety aka scale	0.65 (± 0.18)	0.65 (± 0.20)	0.63 (± 0.21)

As you can see, none of these techniques provides encouraging results, they all decrease the performance of the model over the baseline (without augmentation). This can probably be explained because of the nature of our data which makes augmentation a lot less straightforward than in most ML tasks.

4 Model training

Training with a learning rate of 10^{-3} or 10^{-4} seems to work with most models, see 1.1.2 Random Search.

4.1 Batch training

With CNNs, we need the input data to be equally long (cf. 1 Model architecture). This allows for a batch training, thus I trained the model on batches of the size of the training and test set respectively, regardless of the number of subjects in it. It divided the training time by 5 on the

[4] on the spiral task

CNN50-baseline. However, as I've read [add ref], the model is less efficient as the loss is averaged thus less relevant. See figures below.

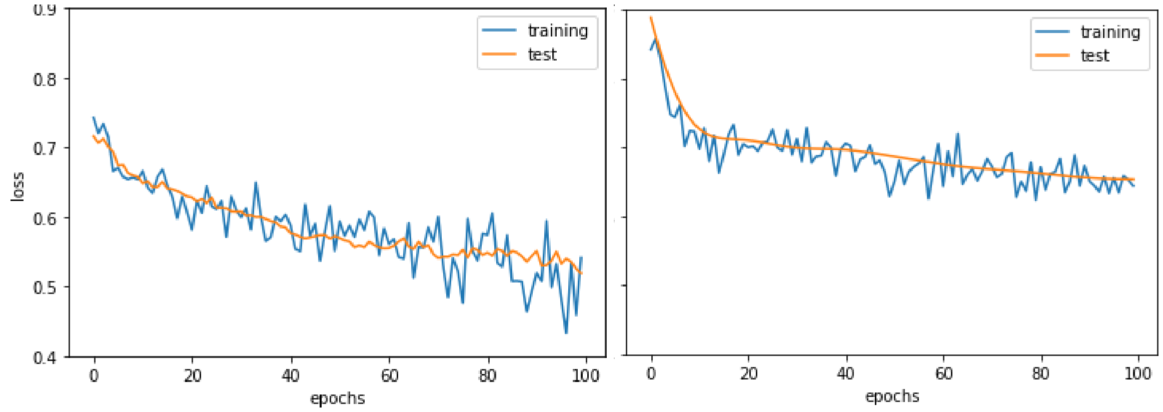
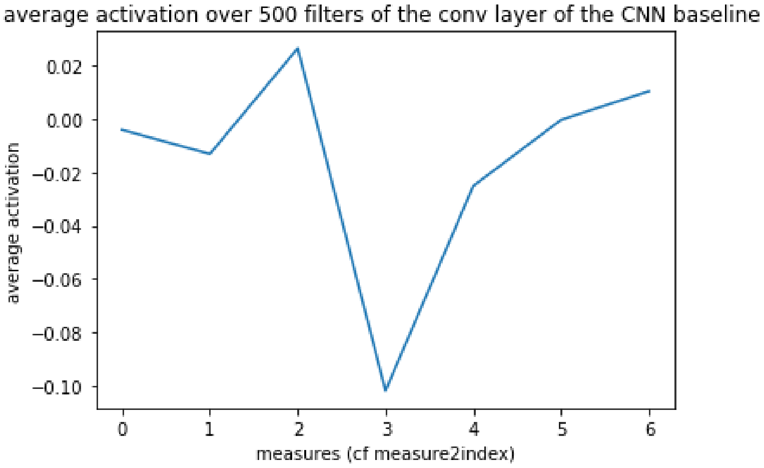


Fig. CNN50-baseline, left : training on a single subject batch, right : training on a multiple subjects' batch.

5 Visualization & Interpretation

5.1 Convolutional Layer

5.1.1 Measures



With the CNN500-baseline, after training on the last fold for 100 epochs (achieving 83% accuracy on the last epoch) we can see that, in average, the neurons corresponding to the button_status measures are less activated than other. This is probably because most of button status are at 1 (i.e. on-paper) for the spiral task we trained on, thus there's no need for 500 filters to monitor it. Moreover, this measure is redundant with pressure. This could also be explained by the binary nature of the variable, it contains less information then other measures. The shape of the plot is similar with the hCNN on the *l* task which suggests that my first hypothesis is wrong.

As the classification accuracy suggests, 500 seems to be a too big number of filters, several of them seems to be monitoring the same thing :

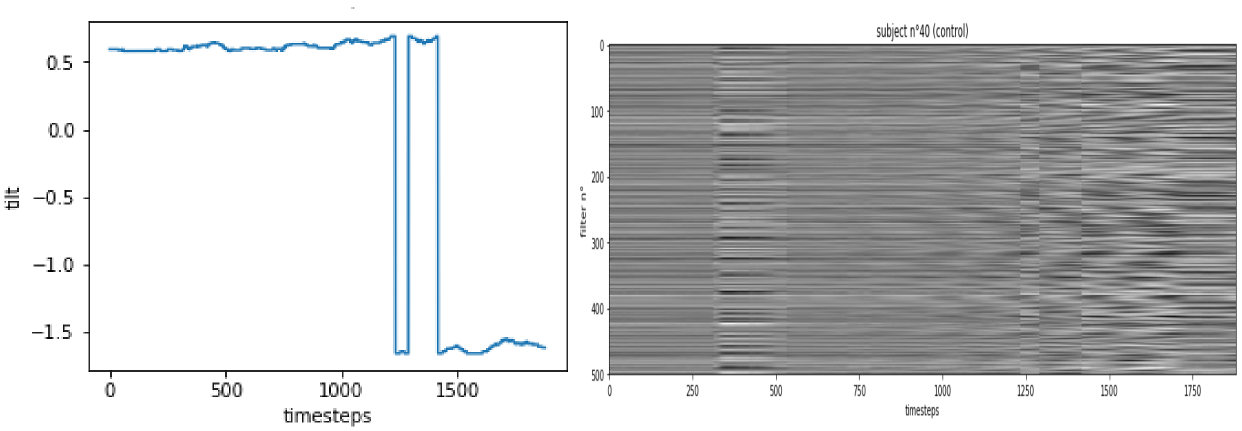


Fig. Subject n°40 (Control) Left : Tilt vs time, right : after a forward pass in the convolution layer (white means high activation, black low).

As you can see in the figure above, a lot of neurons seems to be highly activated when there's a drop in the tilt, see below for a zoom.

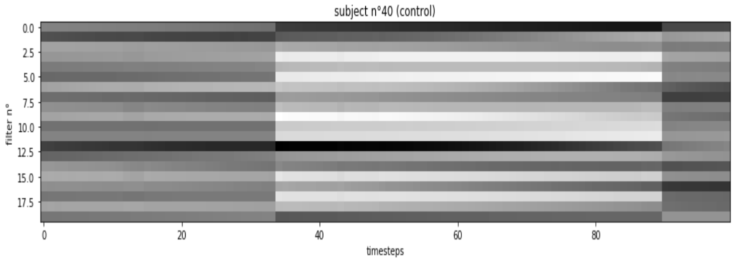


Fig. Zoom of the above figure : feature activation on filters 400 to 420 and timesteps 1200 to 1300.

As you can see, several filters activate at the exact same time : when there's a drop in the tilt measure (this has also been observed with subject n°8 (PD)). See below for details of filters 403 and 405.

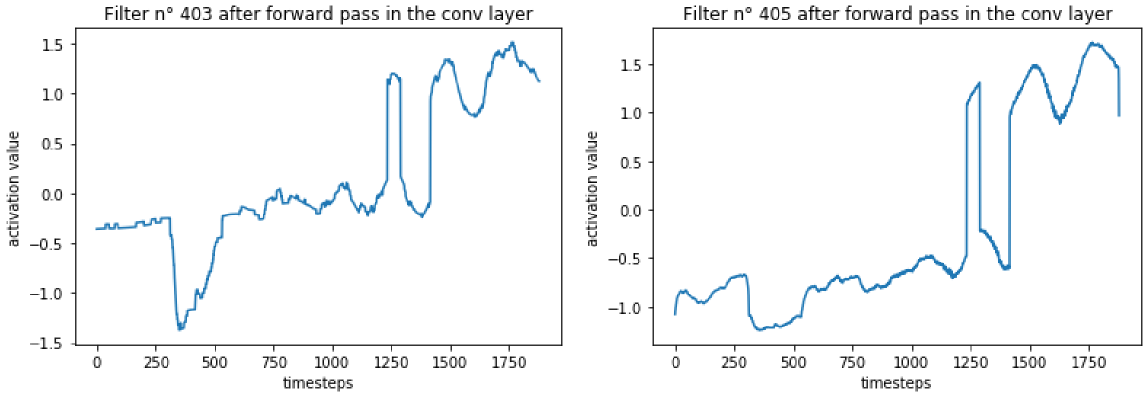


Fig. Activation of filters 403 and 405 (left and right, respectively).

Similar observations can be made when there's drop in other measure's, e.g. pressure or button_status (see average activation plot). Some filters seems to monitor different measure at the same time, see below.

With the Random-Search-CNNs with 2 layers (cf. 1.1.2 Random Search) which had only 4 filters in the first conv layer, none of these 4 filters seemed to monitor only one measure like above but several measures like below. This suggests that interaction between the measures is crucial.

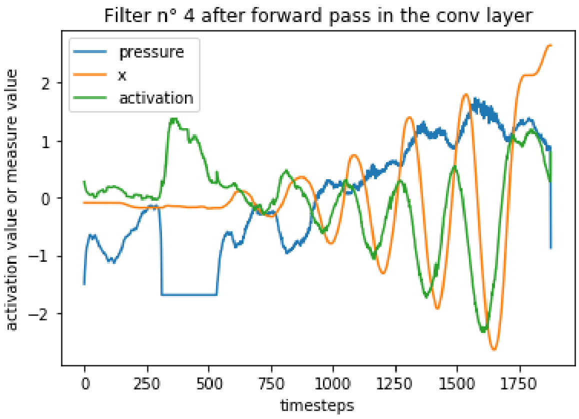


Fig. Filter 4 plotted along pressure and x-coordinate. We can see that the filter first activates when there's a drop in the pressure but then seems to follow the x coordinate.

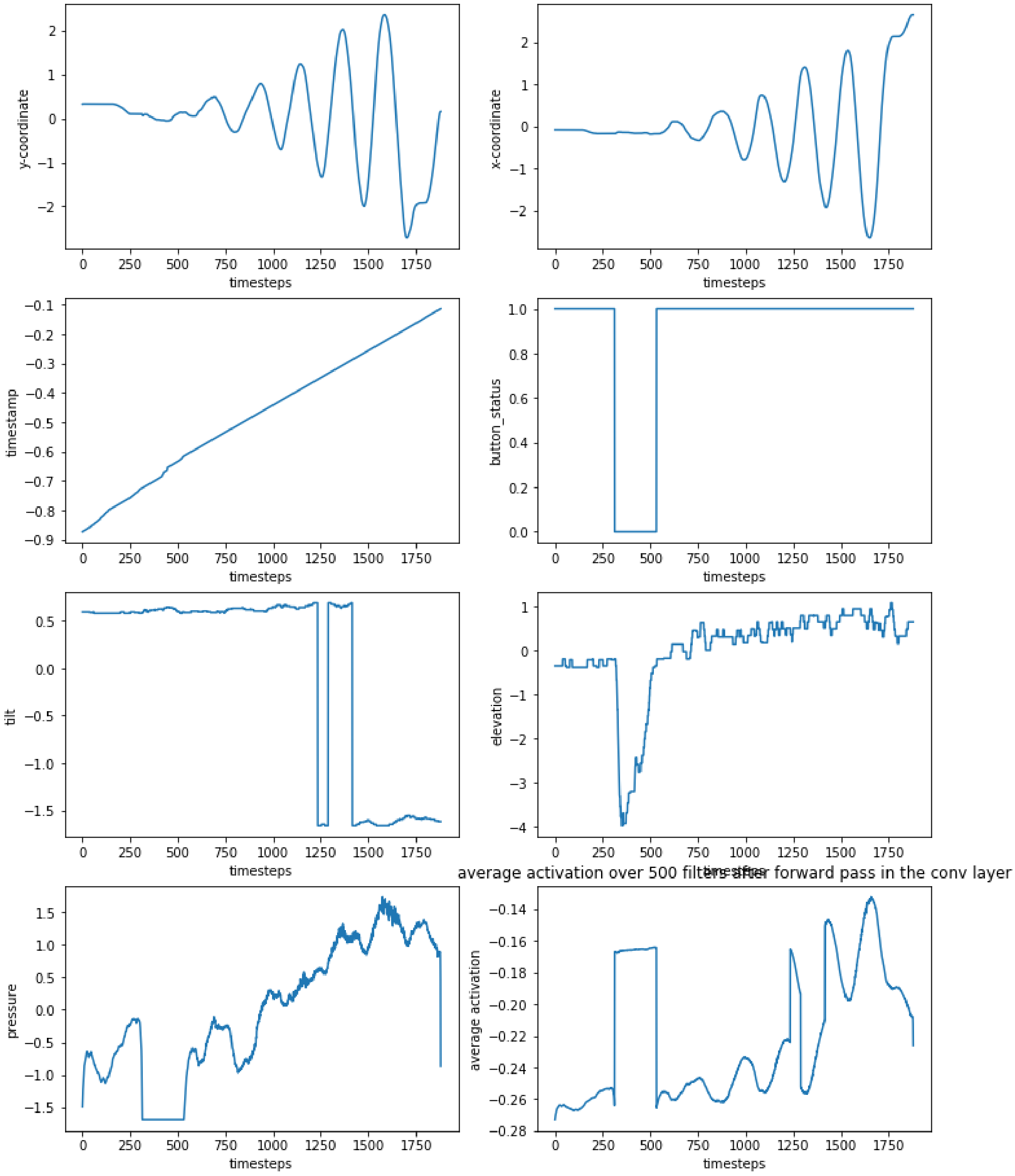
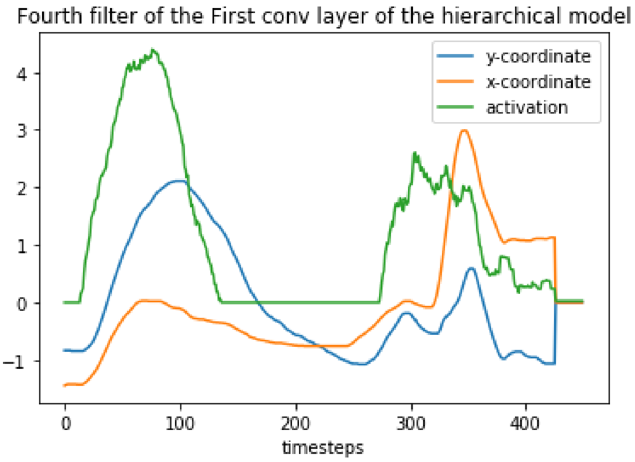


Fig. Subject's 40 (PD) measures vs time and average activation after forward pass in the conv layer (bottom right). We can see extremums in the average activation that match extremums in the measures (e.g peak around 1250 timesteps that we have observed above).

The first conv layer of the hCNN provides similar insights but the second layer is more difficult to interpret as it's more abstract.

5.1.2 on-paper and in-air strokes

With the hierarchical model on the l task, even on folds with good classification accuracy, the neurons seem to be monitoring the same thing regardless of in-air or on-paper strokes. See the plot below.



Plot. Fourth filter seems to monitor both x and y coordinate. The stroke is on-paper until timesteps ~300.

This suggests :

1. training separate models on in-air and on-papers stroke might be useful
2. on the contrary, we don't need to differentiate in-air and on-paper strokes and similar conjectures can be made from both (unlikely).

Conclusion

Deprecated Conclusion - Todo List of 06.06 (keep for reference)

These first experiments with CNNs provide encouraging results : we now outperform Moetesum et al. on the l task and Drotar et al. on the spiral task with 2 different models, respectively. Further experiments are required.

I think I should probably discuss and compare my results with Catherine. It should be interesting to compare her model (CNN2d that takes as input Gramian angular field or spectrogram) and mine. By the way I wonder if she knows the work of Afonso et al. and why motivated her use of Short Time Fourier Transform over Discrete wavelet transform.

I also wanted to extract some features from the data, e.g. speed. On June 18th we decided that it was better to focus only on deep learning for now.

We should discuss about it, try to synthesize my work since the beginning.
I'd like to explore transfer learning methods (coming from online HaW recognition ?). I'm motivated because Moetesum et al. didn't fine-tune their model and just used AlexNet (trained on ImageNet) convolutional layers (cf. report #2).

References

Chiu, J. P., & Nichols, E. (2016). Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, *4*, 357-370.

Granell, E., Chammas, E., Likforman-Sulem, L., Martínez-Hinarejos, C. D., Mokbel, C., & Cîrstea, B. I. (2018). Transcription of spanish historical handwritten documents with deep neural networks. *Journal of Imaging*, *4*(1), 15.

Le Guennec, A., Malinowski, S., & Tavenard, R. (2016, September). Data augmentation for time series classification using convolutional neural networks.

LeNail, (2019). NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software, 4(33), 747, https://doi.org/10.21105/joss.00747

Peddinti, V., Povey, D., & Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In Sixteenth Annual Conference of the International Speech Communication Association.

Salamon, J., & Bello, J. P. (2017). Deep convolutional neural networks and data augmentation for environmental sound classification. IEEE Signal Processing Letters, 24(3), 279-283.

Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., & Van Gool, L. (2016, October). Temporal segment networks: Towards good practices for deep action recognition. In European conference on computer vision (pp. 20-36). Springer, Cham.